Resilient Information Architecture Platform for the Smart Grid

Gabor Karsai, Vanderbilt University

Network Optimized Distributed Energy Systems (NODES) Annual Review Meeting

Feb 12-13, 2019
Project Summary

**Goal:** To create an open source *software platform* to run Smart Grid applications and demonstrate it through *selected applications*. A software platform defines:

- Programming model (for distributed real-time software)
- Services (for application management, fault tolerance, security, time synchronization, coordination, etc.)
- Development toolkit (for building and deploying apps)

**Uniqueness:**

- Focus on distributed *applications* - not only on networking
- Focus on *resilience* – services for fault recovery
- Focus on *security* – maintain confidentiality, integrity, availability
Project Summary

Example Power System: IEEE 30 bus system

Control Room

RIAPS Node:

RIAPS
Computing Platform
Sensors
Network
Actuators
Project Summary

- **Challenges:**
  - How do we build *distributed fault tolerant smart grid applications* in a real-time context? – *It is more than a middleware or networking problem.*
  - How do we *manage* accidental complexities in the development process? – *Developers need tools to be productive.*

- **Deliverables:**
  - Software platform run-time: middleware and other libraries + services used by apps
  - Development toolkit for building, deploying, and managing apps
  - Example applications for the Smart Grid
  - Design documentation
Key outcomes:

- The open source platform will enable developers – sanctioned by utilities - to build reusable components and applications
- The platform specification and its prototype implementation is open source, but for-profit entities will provide software development services for it
- A new open standard that will change how software for the smart grid is developed

Websites:

- https://riaps.isis.vanderbilt.edu/ - Project
- https://github.com/RIAPS - Code base
- https://riaps.github.io/ - Documents
- https://www.youtube.com/channel/UCwfT8KeF-8M7GKhHS0muawg - Youtube channel
Team

- Organizations: Vanderbilt University, North Carolina State University, Washington State University
  - Vanderbilt University (Prof. Gabor Karsai & Abhishek Dubey) Institute for Software-Integrated Systems has decades of experience in researching and developing middleware, model-driven development tools, real-time fault diagnostics, software platforms.
  - North Carolina State University (Prof. Srdjan Lukic, David Lubkeman) is home to the NSF Future Renewable Electric Energy Delivery and Management (FREEDM) ERC, have expertise in power grid based on power electronics, high bandwidth digital communication, and distributed control, including testing of experimental and commercial microgrid controllers.
  - Washington State University (Prof. Anurag Srivastava, Chen-Ching Liu, Dave Bakken) has expertise in power system operation and control, hosts the Smart Grid Testbed (SGDRIL), does research on power systems operation in extreme scenarios, Smart City Testbed, and on fault tolerant computing and middleware for power systems.
Team

- While all team members have electrical engineering background they specialize in complementary fields:
  - Distributed Real Time Embedded systems (Karsai and Dubey)
  - Fault Tolerant Computing (Bakken and Karsai)
  - Electrical Power Engineering (Srivastava, Liu and Lukic)
  - Cyber-Physical Testbeds (Srivastava, Liu and Lukic)
  - Control Engineering (Srivastava, and Liu)

- The team members have solid industry connections that will enable technology transition:
  - Help from industry advisory board to target the technology for the market
  - Ability to do hardware in the loop testing allows having product ready for field installation
## Project Progress

<table>
<thead>
<tr>
<th>PE</th>
<th>Year1</th>
<th>Year2</th>
<th>Year3</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Analysis, Design, Documentation</td>
<td>On track</td>
<td>On track</td>
<td>On track</td>
</tr>
<tr>
<td>3</td>
<td>Platform Services: Detailed Design, Implementation and Verification</td>
<td>On track</td>
<td>On track</td>
<td>On track</td>
</tr>
<tr>
<td>4</td>
<td>Development toolchain design, implementation, verification</td>
<td>On track</td>
<td>On track</td>
<td>On track</td>
</tr>
<tr>
<td>5</td>
<td>Representative Applications Development and Evaluation</td>
<td>On track</td>
<td>On track</td>
<td>On track</td>
</tr>
<tr>
<td>6</td>
<td>Technology Transition</td>
<td>On track</td>
<td>On track</td>
<td>On track</td>
</tr>
<tr>
<td></td>
<td>Demonstrations</td>
<td>On track</td>
<td>On track</td>
<td>On track</td>
</tr>
</tbody>
</table>
Validation Plan - Summary

- **HIL system**
  - For RIAPS (software platform):
    - Development platform: Linux
    - Target platform: Beaglebone Black (embedded ARM)
  - For RAS (WSU):
    - Simulation: RTDS + Target platform (BBB)
  - For Microgrid control (NCSU):
    - Simulation: Opal-RT + Target platform (BBB) + DSP

- **Managed DERs**:
  - RAS: Wind farm
  - Microgrid: PV, Batteries (via inverter)

- **Test plan**
  - Platform: M1.2.1, 1.5.1, 1.7.1, 1.9.1., 2.1.2/4, 2.2.1/3, 2.3.3, 3.1.2/3/5, 3.2.2/4, 3.3.4, 3.4.2, 4.1.3, 4.2.1-3, 4.3.1/3, 4.1.3
  - RAS: 5.8.1, 5.9.2/3, 5.10.1, 5.11.1/2, 5.12.1, 5.12.2
  - Microgrid: 5.2.1, 5.3.1, 5.4.1, 5.5.1, 5.6.2

- **Field validation test sites**:
  - Discussions with IAB members
  - VMWare/LF Energy

- **Large scale simulation plan**: RTDS + Opal-RT
Vision
Distributed Computing for the Grid

Communication Network

Example Power System: IEEE 30 bus system

Control Room

RIAPS Node:

RIAPS
Computing Platform
Sensors
Actuators

Network
I/F

Generators
Synchronous Condensors

GLEN LYN
KUMIS
HANCOCK
NOANOKE

30
29
28
27
26
25
24
23
22
21
20
19
18
17
16
15
14
13
12
11
10
9
8
7
6
5
4
3
2
1
RIAPS
The Software Platform

<table>
<thead>
<tr>
<th>Applications</th>
<th>Component Framework</th>
<th>Platform Managers</th>
</tr>
</thead>
<tbody>
<tr>
<td>Remedial Action Scheme</td>
<td>Component Interactions</td>
<td>Application Manager</td>
</tr>
<tr>
<td></td>
<td>Component Messaging</td>
<td>Coordinator Manager</td>
</tr>
<tr>
<td></td>
<td>Component Scheduling</td>
<td>Discovery Manager</td>
</tr>
<tr>
<td></td>
<td>Event/Time-triggered</td>
<td>Time Manager</td>
</tr>
<tr>
<td></td>
<td>Resource Management</td>
<td>Resource Manager</td>
</tr>
<tr>
<td></td>
<td>Resource Management Service</td>
<td>Device Manager</td>
</tr>
<tr>
<td></td>
<td>Fault Management</td>
<td>Security Manager</td>
</tr>
<tr>
<td></td>
<td>Fault Management Service</td>
<td>Persistence Manager</td>
</tr>
<tr>
<td></td>
<td>Logging</td>
<td>Log Manager</td>
</tr>
<tr>
<td></td>
<td>Logging Service</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Persistence</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Persistence Service</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Security</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Access Control</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Secure Communications</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Secure Information Flows</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Lifecycle Management</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Initialize, Start, Stop,</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Checkpoint, Destroy</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Language Run-time</td>
<td></td>
</tr>
<tr>
<td></td>
<td>C/C++, etc.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Security</td>
<td></td>
</tr>
<tr>
<td></td>
<td>OS Kernel</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Hardware Platform</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Device Interfaces</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Sensors/Actuators/Communications/GPS/...</td>
<td>Network Interface(s)</td>
</tr>
</tbody>
</table>
**RIAPS apps**

**Components + Actors**

Components are the building blocks: defined interfaces (ports) + execution semantics – simple code, may encapsulate complex applications (e.g. numerical solvers) in Python/C++

Actors are built from components that interact solely via messages and are deployed on computing nodes in a network. All applications are built as a fabric of interacting components
RIAPS services Deployment

- RIAPS nodes and apps
  - are managed by a system operator (control room)
  - can join and leave the network at any time
RIAPS services
Discovery

- RIAPS components form a peer-to-peer network, organized and configured via the Discovery service
  - Service provider – service client match-up
**RIAPS services**

**Fault management**

- **Assumption**
  - Faults can happen anywhere: application, software framework, hardware, network

- **Goal**
  - RIAPS developers shall be able to develop apps that can recover from faults anywhere in the system.

- **Use case**
  - An application component hosted on a remote host stops permanently, the rest of the application detects this and ‘fails over’ to another, healthy component instead.

- **Principle**
  - The platform provides the *mechanics*, but app-specific behavior **must be** supplied by the app developer
- **Group membership:**
  - An app component can dynamically create/join/leave a *group* that facilitates fast communication among members

- **Leader election:**
  - A group can ‘elect’ a *leader*: a component that makes global decisions. Election is automatic and fault tolerant, group members directly interact with the leader.

- **Consensus:**
  - Group members can ‘vote’ in a *consensus* process that reaches agreement over a value.

- **Time-coordinated control action:**
  - Group members use a combination of the above three features to agree on a *control action* that is executed at a scheduled point in time in the future

- **Application example – Microgrid control**
  - Group Membership and Leader Election: ‘microgrid’ groups for sharing information for better control
  - Consensus: on voltage and frequency values
  - Time-coordinated control action: microgrid to islanded mode
Secure applications

- Application packages are compressed, encrypted and cryptographically signed before deployment. The recipient nodes verify cryptographic signatures, decrypt and install the app.

- All app-level communications are protected by the CurveCP (elliptic curve encryption) on the messaging layer: ZeroMQ. All communications are protected via public/private keypairs, that are generated dynamically when the app is deployed. Keys are installed whenever an app-level network connection is established, and they are part of the deployment package, stored in a certificate store on the target nodes.
Secure messaging between services

- Unsecured – communication is among processes on the same host
  - Deployment service $\leftrightarrow$ actor
  - Deployment service $\leftrightarrow$ discovery service
  - Actor $\leftrightarrow$ discovery service

- Discovery service
  - OpenDHT already encrypts all service registrations
  - Discovery service instances use a single shared key across the network
  - Private key on node is protected via file access control
RIAPS Security
Application level protection

- **Network threats**
  - Each app actor is allowed to accept network packets only from hosts participating in the same app
    - App-level firewall on the incoming messages

- **Insider threats (malicious / flawed app)**
  - Network protection
    - App’s view of the network is explicitly modeled and used in configuring firewalls on the hosts
    - Firewall allows only communication within the RIAPS app’s network (both direction)
    - Exceptions are configurable by system integrator (‘owner’)
  - Information flow protection
    - AppArmor (a Linux Mandatory Access Control [MAC]) system is used to constrain the app’s access
    - Security profile is enforced by the trusted installer (Deployment Manager)
    - Default access: own files, core system packages, TCP/UDP protocols – very constrained – maybe necessary to allow app-specific overrides
## RIAPS Security Test plan

<table>
<thead>
<tr>
<th>Category</th>
<th>Layer</th>
<th>Feature</th>
<th>Test for</th>
</tr>
</thead>
<tbody>
<tr>
<td>Secure Information Flows</td>
<td>Transport (ZeroMQ)</td>
<td>Key management</td>
<td>Key distribution for app communications</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Uniqueness of keys</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Authentication</td>
<td>Connection can’t be created without a proper key</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Encryption</td>
<td>No app information is shared in cleartext</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Integrity</td>
<td>No tampering with the message</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Denial of service</td>
<td>Resilience against typical DoS attacks (SYN flooding)</td>
</tr>
<tr>
<td></td>
<td>Discovery (OpenDHT)</td>
<td>Key management</td>
<td>Key distribution for discovery service</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Authentication</td>
<td>Discovery service cannot be used without proper key</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Encryption</td>
<td>Unauthorized parties cannot access discovery</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>All discovery messages are encrypted</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Denial of service</td>
<td>Resilience against typical DoS attacks (UDP flooding)</td>
</tr>
<tr>
<td>Application-level Security</td>
<td>Application management</td>
<td>Node configure</td>
<td>Node is properly configured</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Credentials</td>
<td>Application credentials are created</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Profile</td>
<td>Deployment service installs proper AppArmor profile</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Access Control</td>
<td>App can access only those services that are enabled by the profile</td>
</tr>
</tbody>
</table>
RIAPS Development Tools

For details please see: https://github.com/RIAPS/riaps-dsml
Objective: To minimize wind curtailment while keeping the system reliability

- Constraints: Wind farm operational limits, line limits
- Distributed Implementation: Distributed Simplex Method in Linear Programming
- Offline simulation with N-1 computational block failure
- Verification with decentralized algorithm implemented in real time using CISCO Fog and Beaglebone Black

<table>
<thead>
<tr>
<th>Mode of Operation</th>
<th>Wind Generation Curtailment</th>
<th>Execution time</th>
</tr>
</thead>
<tbody>
<tr>
<td>All nodes operational</td>
<td>100.0 MW -&gt; 84.14 MW</td>
<td>0.956s</td>
</tr>
<tr>
<td>Node failure</td>
<td>100.0MW -&gt; 81.60 MW</td>
<td>1.035s</td>
</tr>
</tbody>
</table>

- Without RAS, one of the transmission lines is overloaded by 16.6%.
- RAS algorithm curtail wind farm outputs and totally eliminated line overflows violation without any load shedding.

Application
RAS-1: 179 Bus Case with Idaho Power

Substation #1

Substation #2

Substation #3

179 Bus WECC case

4 BBs/ 4 CISCO Fog

GPS → PMU → Actuator

PMU → Measurements

Control Commands → Actuator

GPS → PMU → Actuator

PMU → Measurements

Control Commands → Actuator

GPS → PMU → Actuator

PMU → Measurements

Control Commands → Actuator

Fog/ BeagleBoard With RIAPS

Fog/ BeagleBoard With RIAPS
Application
RAS-2: Optimal Under-frequency Load Shedding

Real-time Simulation Architecture

Simulation Control

Power System Real-Time Simulation

RIAPS Development Control Station

VM VirtualBox

RTDS (Real-Time Digital Simulator)

Measurements (e.g., frequencies and voltages)

Load shedding decisions

Agents in RIAPS Nodes
Case Study

Modified IEEE 14-Bus System

Contingency Scenario
- The power grid splits into two islands due to the tripping of three lines and two transformers;
- The South Island is with an active power deficit.

Simulation Results
- (RTDS + 3 RIAPS nodes)
- Without RAS, frequency drops quickly leading to the system collapse;
- With RAS-2, the frequency decline is stopped at \( t = 3.04 \) sec.

<table>
<thead>
<tr>
<th>UFLS Schemes</th>
<th>With RAS-2</th>
<th>No RAS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Active Load Shedding (MW)</td>
<td>19.37</td>
<td>66</td>
</tr>
<tr>
<td>Reactive Load Shedding (MVar)</td>
<td>9.58</td>
<td>32</td>
</tr>
<tr>
<td>Lowest Frequency (Hz)</td>
<td>58.2</td>
<td>System collapse</td>
</tr>
<tr>
<td>Stable Frequency (Hz)</td>
<td>58.89</td>
<td>System collapse</td>
</tr>
<tr>
<td>Time when Frequency Decline Stops (SEC)</td>
<td>3.02</td>
<td>System collapse</td>
</tr>
<tr>
<td>MW Reduction in Load Shedding Compared with “No RAS”</td>
<td>70.65%</td>
<td>N/A</td>
</tr>
<tr>
<td>MVAR Reduction in Load Shedding Compared with “No RAS”</td>
<td>70.06%</td>
<td>N/A</td>
</tr>
</tbody>
</table>
The blue curve is the NERC under-frequency load shedding performance and modeling curve.

The orange curve is the frequency response with RAS2.

As the orange curve is above the blue curve, the NERC requirements are met.
Main challenge: provide stable operation in all operating modes

Basic functionalities can be adapted to any microgrid topology (e.g. nested microgrids with moving boundaries).

Focus on time-sensitive applications

Energy management algorithms not the focus
• Yuhua Du, Hao Tu, Srdjan Lukic “Distributed Control Strategy to Achieve Synchronized Operation of an Islanded MG” Accepted to *IEEE Transactions on Smart Grid* (IEEE Early Access)

• Yuhua Du, Xiaonan Lu, Jianhui Wang, and Srdjan Lukic “Distributed Secondary Control Strategy for Microgrid Operation with Dynamic Boundaries” submitted to *IEEE Transactions on Smart Grid* (second review)
Microgrid Control System
Advanced HIL Testbed

- Current work: Building the IEEE 34-bus system in Opal with unbalanced loads
- Implementing Distributed Secondary Control Strategy for Microgrid Operation with Dynamic Boundaries in RIAPS
Primary Control implemented on Texas Instruments F28377S MCU
Inverter power stage simulated in Opal FPGA (switching model, step time 0.5µs); microgrid simulated on CPU (100µs)
Custom-designed board allows for four DERs to be emulated
Next steps: (1) Increase system size to 10+ nodes (2) Implement dynamic boundaries algorithm in HIL microgrid
Moving towards a hardware implementation using FREEDM solid-state transformer (SST) hardware testbed

Testbed consists of 3 SSTs, 3 ESS, smart house, house loads, programmable loads, etc.

NCSU integrated RIAPS+MCU platform with SST and ESS power electronics hardware
Project Challenges

- **Past challenges:**
  - Many milestones required production of significant documentation → Focus on the essentials in documents
  - Complexity of software code based required for the platform is considerable → Use existing open-source packages (after careful analysis and testing)

- **Possible project challenges going forward:**
  - Testing and validation of the platform *outside of lab* will be a significant effort → Explore opportunities for field testing
  - Completion of documentation, including training materials → Prioritize
List of Achievements
https://riaps.isis.vanderbilt.edu/papers.html

RIAPS


- P. Volgyesi, A. Dubey, T. Krentz, I. Madari, M. Metelko, G. Karsai, "Time Synchronization Services for Low-Cost Fog Computing Applications", International Symposium on Rapid System Prototyping (RSP), Seoul, South Korea, 10/2017


List of Achievements

https://riaps.isis.vanderbilt.edu/papers.html

RAS


List of Achievements
https://riaps.isis.vanderbilt.edu/papers.html

Microgrid

Tech to Market Path and IAB

- **Objective**
  - Open-source platform, supported by a spin-off

- **Market segment**
  - Power system software developers and users

- **Commercial partners/advisors**
  - ABB, Cisco, Duke Energy, National Instruments, RTE France, National Grid, OSISoft, Siemens, South California Edison, TVA

- **Activities**
  - Project is open-sourced ([https://github.com/RIAPS](https://github.com/RIAPS), Apache License)
  - Collaboration with Linux Foundation/Energy Foundation
    - RIAPS is an LFEnergy project ([https://www.lfenergy.org/](https://www.lfenergy.org/))
    - Exploring field test opportunities with industrial partner
      - VMWare/Palo Alto microgrid effort
Plans

Next steps:

– Completion the testing of security features
– Completion of documentation + training materials
– Complete improved distributed applications (RAS, microgrid control, transactive energy, etc.)

Interactions:

– Started discussions with VMware about potential field testing and technology exchange